Verifiable and Resource-Aware Component Model for IoT Devices

Arthur Gatouillat, Youakim Badr

Verifiable and Resource-Aware Component Model for IoT Devices

Arthur Gatouillat
Univ Lyon, INSA-Lyon
LIRIS, UMR5205
F-69621, France
arthur.gatouillat@insa-lyon.fr

Youakim Badr
Univ Lyon, INSA-Lyon
LIRIS, UMR5205
F-69621, France
youakim.badr@insa-lyon.fr

ABSTRACT
Most connected objects feature very limited capabilities that present challenges in terms of data processing and connectivity. In addition, heterogeneity of smart Internet-of-Things devices also causes interoperability problems. These limitations lead to strong hardware and software constraints that must be considered as early as possible during the design process. In this paper, we introduce a smart object component-based model to build complex smart objects by composition mechanisms in a similar way to Web service compositions. The smart object model extends artifact types and describes its structure and behavior in terms of attribute value pair, state-based lifecycle and services. Moreover, we propose a formal specification based on the intuitive multiplicative segment of intuitionistic linear logic not only to express consumable resources but also to automate composition from logical proofs.

CCS Concepts
• Computer systems organization→Embedded and cyber-physical systems • Software and its engineering→Software system models • Software and its engineering→Formal methods

Keywords
Linear logic; formal verification; connected objects composition.

1. INTRODUCTION
By 2022, there will be as many as 29 billion Internet-enabled connected objects [6] that will fall under the Internet-of-Things (IoT) label. The assertion behind the Internet of Things is to enable the interconnection of virtually any objects from the physical world without any human intervention. Connected things, also referred as smart objects, cover a wide spectrum of sensors, actuators, robots, unmanned vehicles or even wearable devices. Nowadays, a wide variety of smart objects is available for a very low price, thus unlocking unlimited opportunities to build advanced smart services available to virtually any external tier. The Internet-of-Things, thanks to seamless connection of people, processes, data and things, is bringing considerable business and innovation opportunities. However, the current technical state of the IoT is still in its infancy and encounters various challenges and limitations that still prevent the creation of true wide-scale Internet-of-Things applications. One of these challenges is brought by the unavailability of conventional programming paradigms and software engineering processes, that cannot be used for IoT applications because of the limited resources embedded in traditional IoT devices, along with the hardware and software heterogeneity of such devices. In order to build full scale IoT solutions, these challenges must be studied simultaneously:

Limited resources. The scope of what can be considered as a connected object is wide, and it ranges from small, battery-operated, and computationally limited sensors to continuously powered complex systems featuring numerous sensors and actuators with greater computing potential. However, most connected objects feature very limited capabilities that present challenges in terms of data processing and connectivity. The limitations typically come in terms of computing power (i.e., the embedded CPU frequency does not usually exceed a few tens of megahertz), of live memory (i.e., available RAM memory of a few tens of kilobytes), and in terms of storage (usually not exceeding a few hundreds of kilobytes). These limitations lead to strong hardware and software constraints that must be considered as early as possible during the design process, in order to guarantee that higher-level systems requirements can and will be achievable using resources limited connected objects.

Interoperability. Heterogeneity of smart Internet-of-Things devices also causes interoperability problems. Indeed, the proliferation of wireless communication standards (e.g., IEEE 802.15.4, WiFi, BLE, …), communication protocols (e.g., CoAP, LORA, MQTT, …), or cellular communication technologies (e.g., GSM, UMTS, LTE, …) leads to potential interoperability issues. This is an element of concern when integrating a wide range of smart objects, and it makes their reusability challenging. Despite many recent initiatives, the emergence of standards remains highly fragmented, leading to divergence in vocabularies, methods and models (OneM2M, IoT reference architecture, etc.). The design of interoperable smart objects remains balkanized without an integrative approach to make real progress in reducing software, hardware and communication heterogeneity.

In order to empower users with capabilities to make their own decisions regarding their smart object data and services, we propose a holistic approach that attempts to balance the limitation of resources with model genericity by proposing:

1) a reusable component-based model as a building block for specifying smart objects and their cyber-physical properties, and;
2) a resource aware verification framework for connected objects.

Our reusable component-based model makes possible to build smart objects "as a combination" of sensors, controllers, actuators, physical things, people and consumable resources. The "logic of combinations" already appears in the fields of software engineering (service composition) and business services (service bundling). The composition mechanism aims at quickly creating software applications from existing software components rather than

MEDES '17, November 7–10, 2017, Bangkok, Thailand

ACM ISBN 978-1-4503-4895-9/17/11…$15.00
https://doi.org/10.1145/3167020.3167056

building them from the ground up. However, our component-based model for a smart object differs from current component-based software (Web services, Enterprise JavaBeans) in several important ways. First, it integrates the cyber and physical domains into a cohesive unit. Secondly, it includes information models, services and a lifecycle characterization to describe a smart object structure and behavior and how it evolves through its lifecycle to achieve desired user-set goals. The smart object component-based model is enabled with simple translation into verifiable linear logic statements, which can then be then used to verify the overall composition, resulting in better global system reliability.

The remaining paper is organized as follows: section 2 quickly introduces a motivating example and provides details about resource aware framework, while section 3 introduces the linear logic-based verification part for specifying and composing smart objects. Section 4 describes related works and highlights existing contributions in regard to our contribution. Eventually, section 5 concludes our work and identifies further research directions and challenges.

2. A COMPONENT-BASED MODEL FOR SMART OBJECTS

2.1 Motivation Case-Study

We consider home-automation for e-Health care as a motivation case-study. A patient is equipped with wearable devices performing continuous monitoring of their various physiological signals such as cardiac and respiratory activity, body-core acceleration and skin resistivity. The patient's smart-house embeds a variety of connected objects and smart devices, enabling the adaptation of the ambient environment and maintaining a safe and high quality-of-life standard. Devices such as thermostats, blinds and windows controllers, light bulbs, controllers and a wide range of actuators make possible to continuously modify the smart home characteristics, and to ensure pleasant living environment.

The smart home in our case-study is equipped with a gateway, which can control and subscribe various hardware services. This gateway is enabled with signal processing capabilities and is continuously powered, thus relaxing the low energy constraint of traditional embedded IoT devices. Such gateway can consequently be equipped with multiprotocol connectivity, and can subscribe to a variety of devices without having to consider protocol-caused interoperability problems.

Our concern is to offer comprehensive modelization, composition and verification framework for all the sensors, actuators and gateways included in the smart-home, in order to achieve a higher-level objective requiring collaboration of connect objects that cannot be implemented using sensors or actuators alone. In our case-study, the composition makes use of physiological sensors and environmental actuators to compute and health estimation of the smart-home occupant that is used to either trigger environmental actions (such as blind opening, door locking, …) or to trigger external response if a critical situation is detected (such as a health crisis).

2.2 Requirements for Reusability

Service computing, including the service-oriented architecture (SOA), web services and service composition mechanisms provide a noteworthy computational paradigm to implement distributed systems by means of agnostic, distant executable and loosely-coupled services [16]. Service computing can handle problems relevant in the context of the IoT in terms of modularity, reusability and scalability. Indeed, these three characteristics are of prime interest in the study of the IoT, as IoT-based systems must be able to manage changes in networked devices and their environment, where devices can be added or removed at any moment. As a result, modularity and reusability become of important features of IoT-based systems. In fact, these systems rely on connected devices that can be reused in various applications and in a wide variety of contexts. For instance, a heart rate monitor smart device can be used to monitor physical activity or for detection of heart malfunction. This leads to two IoT-based applications with different requirements in terms of reliability and resiliency. The heart rate monitor smart device must be as reusable as possible. Finally, the scalability must also be considered when building IoT systems. Since what is traditionally considered as part of the IoT is wide, the number of connected devices in IoT-based systems varies greatly between applications, ranging from home-automation applications to city-wide data collection. As a result, models and design methods for building IoT systems must be able to easily scale up, and handle both smaller-size and massive-size systems.

However, design methods and architectural styles introduced by the service-oriented community such as Web services and Web services composition approaches remain inadequate to design and implement smart objects for several reasons.

Firstly, Web services and other component-based software (e.g., EJB, JBI, …) are pure programs. Despite their ability to define consumable resources and their constraints as non-functional properties, they do not allow for a complete answer to smart objects requirements, because the cyber and physical parts can be represented and taken into account during service composition mechanisms.

Secondly, the primary disadvantage of Web services and other component-based software is the separation between data and their processing. In fact, services are often oriented towards data processing (i.e., functions or tasks) and aim to be reusable in workflow processes where data are defined and manipulated in databases. The separation violates the smart objects modularity and their capability to be self-contained and interoperable.

In order to tackle these limitations, we propose a component-based model for smart objects based on the *artifact type* [13] extended with capabilities enabling the model to consider constraints on functional and non-functional properties, consumable resources and feedback loops. Artifact types, formally known as business artifacts, are entities which combine both data and their manipulation into cohesive and modular units. It includes an informational model, consisting of attribute-value pairs, a set of services that manipulate attributes, and a lifecycle, which describes the evolution of the system over time through service invocation. A component-based model for smart objects based on the *artifact type* extends the artifact type as proposed by Bhattacharya et al. [4] and can be formally defined as the following:

We start by assuming the existence of the following pairwise disjoint countably infinite sets:

- $\mathcal{C}$ of component-based smart object names
- $\mathcal{A}$ of attributes names
- $\mathcal{T}_{\text{prim}}$ of primitive types, including **Boolean**, **Integer**, etc.
- $\mathcal{T}_{\text{com}}$ of complex types, where every element of $\mathcal{T}_{\text{com}}$ is a set of elements of $\mathcal{T}_{\text{prim}}$
- A type is an element in the union $\mathcal{T} = \mathcal{T}_{\text{prim}} \cup \mathcal{T}_{\text{com}}$.

Definition 1: A **component-based smart object** is a tuple (**C**, **A**, **τ**, **L**), where $\mathbf{C} \in \mathcal{C}$ is the smart object name, $\mathbf{A} \subseteq \mathcal{A}$ is the set of attributes, **τ** is the total function that maps **A** to the set of types $\mathcal{T} \cup \mathcal{C}$, **L** is the lifecycle of the artifact class as defined below.

Definition 2: A **component-based smart object lifecycle** is a directed graph with distinguished initial node and final nodes, also

called a transition network. Nodes of the digraph represent states of lifecycles and arcs represent state transitions. We define component-based smart object lifecycle as a tuple $\mathbf{L} = (\mathbf{S}, \mathbf{s_0}, \mathbf{F}, \boldsymbol{\delta})$, where $\mathbf{S}$ is the set of states, $\mathbf{s_0} \in \mathbf{S}$ is the initial state, $\mathbf{F} \in \mathbf{S}$ is the set of final states, $\boldsymbol{\delta}$ is the state-transition relation $\boldsymbol{\delta} \subseteq \mathbf{S} \times \mathbf{S}$. The lifecycle can be defined in two different specifications: 1) as an imperative specification such as directed graph or automaton. 2) as a declarative specification as a set of rules (if-condition-action) or as a guard-stage-milestone (GSM) model. In the next paragraph, we introduce how component-based smart object is described in terms of GSM model.

Definition 3: A **service** is a tuple ($\boldsymbol{v}$, **I**, **O**, **Pre**, **Eff**) where $\boldsymbol{v}$ is the service name, **I**, **O** $\subseteq \boldsymbol{C}$ are the finite sets of input and output smart objects, **Pre** is the set of pre-conditions whereas **Eff** is the set of the conditional effects. **Pre** and **Eff** are formulas written in first-order logic as defined below.

In our use case, both sensors and actuators are abstracted trough a set of what we define as hardware services. Hardware services expose connected object functionalities to the external world as independent and reusable services. They rely on hardware-software interfaces and can be divided into three categories: hardware services describing functional properties, internal resource management services and on-the-fly hardware module configuration services. It is worth to mention that hardware services are exposed through radio communication to external connected devices as members of the set:

$$HS = < actuation, sensing, resources\ management >$$

Definition 4: A service **pre-condition** (**Pre**), is a first order formula composed as conjunction of *defined operator* (***def***($\boldsymbol{C}$, $\boldsymbol{A}$)) and its negation (***Notdef***($\boldsymbol{C}$, $\boldsymbol{A}$)), where $\boldsymbol{C}$ is an component-based smart object and $\boldsymbol{A}$ is an attribute belonging to $\boldsymbol{C}$. Formally, the service pre-condition is the tuple **Pre** = (***Def***, ***Notdef***, **θ**), where ***Def*** $\subseteq \mathcal{A}$ is the set of attributes that should be defined, ***Notdef*** $\subseteq \mathcal{A}$ is the set of attributes that should not be defined, **θ** is the total function that maps the elements of ***Def*** and ***Notdef*** to their corresponding an component-based smart objects in $\boldsymbol{C}$.

Definition 5: A service **effect** (**Eff**) is a first order formula expressed as a conjunction of defined (***def***($\boldsymbol{C}$, $\boldsymbol{A}$)) and new (***new***($\boldsymbol{C}$, $\boldsymbol{A}$)) operators**,** where $\boldsymbol{C}$ is an artifact class and $\boldsymbol{A}$ is an attribute belonging to $\boldsymbol{C}$. Formally, the effect is a tuple **Eff** = (**Def**, **New**, **θ**), where **Def** $\subseteq \mathcal{A}$ is the set of attributes that should become defined when the service terminates its execution, **New** $\subseteq \mathcal{A}$ is the set of attributes that should become defined and refers to newly created component-based smart object instances when the service finishes executing, **θ** is the total function that maps elements of **Def** and **New** to artifact classes in $\boldsymbol{C}$ they belong to.

Definition 6: A Rule is a tuple **r** = (**Evt**, **Cond**, **Act**), where **Evt** ∈ **E** is an event that causes smart objects to react, **Cond** is the condition that must hold before applying any action, **Act** is the action such as executing services or changing states. A rule can be written as:

on *event* `if` *condition* `do` *service*

The artifact-centric approach demonstrates many advantages and benefits, including a natural modularity and componentization of self-contained entities and a framework of varying levels of abstraction aimed at the development of goal-oriented components, instead of the traditional function-oriented components in the case of Web services.

For our use-case, the component-based smart object requires the composition of sensors, actuators, controllers and physical plants, into a standalone (and composite) component, and its artifact-based representation is given in Figure 1. The composition is ensured by the feedback loop and supported by the artifact-based model to include information, rules and services. Feedback loop elements are instantiated in the lifecycle to describe the logic of operation of the overall composition (see Figure 3).

The data model of the connected object is specified using a structured document (which can be specified using languages such as JSON or XML). This data model contains generic attributes such as a universal identifier or all the data relevant to the composite system, such as the respiratory rate value or the heart rate value. It is accompanied with a list of sensors, actuators, controllers and plants.

When it comes to hardware services, the example only briefly details the service acquisition service, associated with the stages describing physical data acquisition. The pre-condition to execute the service is the *ComputePhysicalActivity* variable, while the condition is the triggering of the variables associated to the milestones of the stage. This service does not take any input, and updates the relevant variables included in the data information model to the newly received biomedical data.

Regarding GSM rules, our example only details a single rule, and it describes the assignment of a global variable, *currentStage*, to the name of the executing stage if the *ComputePhysicalActivity* event is triggered. Such global variable can then be exposed to external tiers and be used to follow the composition execution remotely.

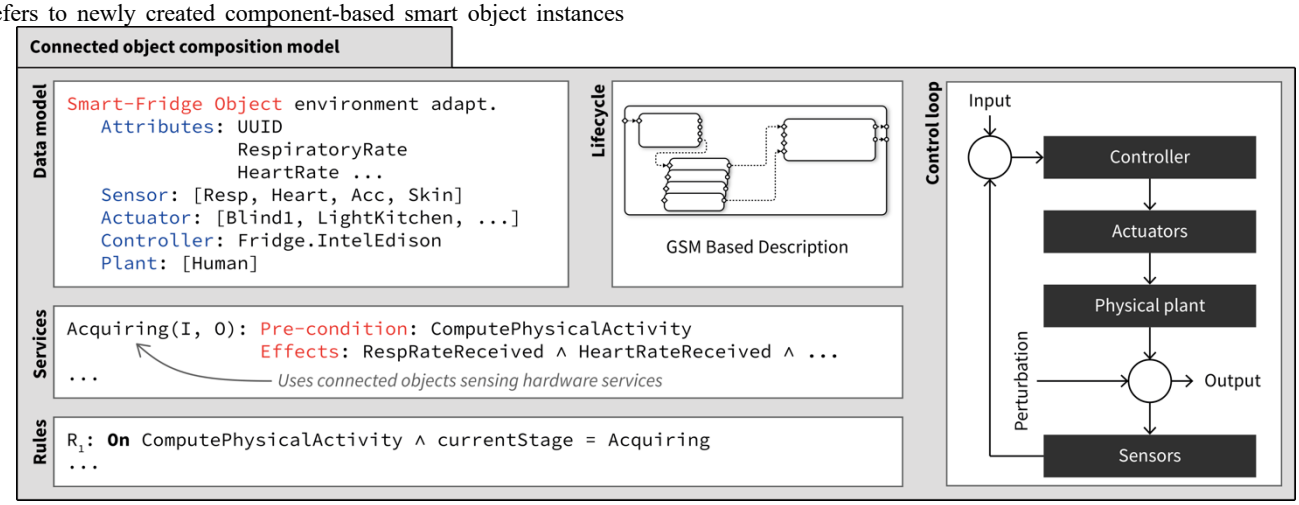

Figure 1. Artifact-based representation of a connected component

The traditional approach for lifecycle description is to use a traditional state-based lifecycle and a set of transition rules that invoke services and results in changing the artifact current state. However, this approach was criticized for its lack of reusability and modularity. Moreover, such lifecycles are often difficult to interpret from a goal oriented standpoint, resulting in an end-user unfriendliness when it is necessary for the users to create their own goals. To tackle this lack of goal-oriented vision, we chose the guard-stage-milestone lifecycle model in our smart object description, and such model is described in the following subsection.

2.3 Smart Object Goal Oriented Lifecycle

The GSM Lifecycle model takes a declarative approach to specify the smart object lifecycles [11], and features better expressivity than state transition diagrams. As graphically depicted in Figure 2, the GSM lifecycle is defined in terms of three components:

- ***The Guards*** are the entry point of a stage and denotes a condition over the information model or an external event expressed as a *sentry*. The stage body is activated when all of the guards are achieved.
- ***The stage body*** contains one or multiple services, with external service invocation capabilities. It also may contain embedded sub-stages. The stage is deactivated when all of its milestones are achieved.
- ***The milestones*** are, in opposition to guards, conditions over the component-based smart object attributes or an external triggering event that deactivated the stage body when all its services are executed and milestones are achieved.

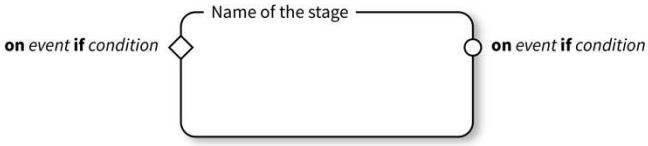

Figure 2. Guard-stage-milestone graphical representation

Both guards and milestones are expressed under the form of a sentry, which is an expression taking the form "**on** *event* **if** *condition*," "**on** *event*" or "**if** *condition*." They consist of triggering event types and/or a condition. Both the triggering events and conditions on attributes and internal or external stages.

A stage can be represented graphically as depicted in Figure 2**Error! Reference source not found.**. The rounded rectangle represents a stage, which may contain sub-stages, the diamond symbol on the left side represents a guard and whereas the circle symbol on the right side presents a milestone. A stage may have one or more guards or milestones. A complete lifecycle consists of the assembly of several stages and sub-stages. The GSM representation smart object lifecycles allows an improved modularity and ease of use because of its declarative and graphical representation (see Figure 3).

2.4 Smart Object Composition

The expressiveness of GSM-based models lies on their modularity, which leads to choose it as a modeling framework for smart objects and investigates a composition mechanism to offer end-users mean to easily, safely and efficiently combine their smart objects and build IoT-based systems.functional perspective, component-based smart objects are seen as compositions or assemblies of hardware services exposed by sensors and actuators. By composition, smart objects achieve a goal that cannot be fulfilled alone with existing sensors and actuators. In our motivation case-study we assume that the patient is equipped with three biomedical sensors: a two-in-one sensor for sensing cardiac and respiratory activities, streaming the heart rate and the respiratory waveform in a real-time. Interacting with the physical world is accomplished with sensors and actuators through their hardware services. The electrodermal activity sensor measures skin resistivity values (as a sensing hardware service) while the body-core acceleration measures continuously heart acceleration rate values. At the infrastructure level, such streaming services are integrated through a publish-subscribe messaging bus (PubSub Bus) such as MQTT[1]. The bus allows the integration of various hardware services through messages or event exchanges. By such, sensors and actuators play the role of publishers to disseminate their values and the role of subscribers to receive their configuration parameters in real-time. All these wearable sensors are battery-powered, and are thus very constrained in terms of processing power and storage capabilities. They also must use low-energy wireless protocols in order to operate with optimal battery life.

As depicted in Figure 1, the smart object, representing an *Ambient Health Care Component*, consists of a data model, services, rules, GSM-based lifecycle and control loop. The information model includes various information such as a smart object's identifier (Universal Unique ID), a list of object-related attribute-value pairs, and aggregates sensors (i.e., electrodermal activity sensor, cardiac and respiratory sensor, body accelerator sensor, …), actuators (i.e., lights, blinds, refrigerator, …), a list of controlled physical plants (i.e., patient body), and controllers (i.e., Intel Edison board to deploy the MQTT PubSub bus), each of which is a standalone smart object.

Figure 3 shows the GSM-lifecycle, which consists of 6 stages, each of which has guards and milestones. The outer stage, named *Ambient Health Care Component*, aims at detecting patient activities in order to adapt the ambient environment accordingly. For example, if the patient is sitting (accelerometer) and their heart beat and skin resistivity are above high-risk threshold, the *Make Smart Decision* stage is activated to infer whether the remote emergence must be informed. Nevertheless, if the patient is practicing his/her sport exercises and the heart rate is accelerated, the blinds are opened during the summer.

As illustrated in Figure 3, the first stage, called "Acquiring physical data," represents the subscription of the ambient health care smart object to the streaming sensing hardware services offered respectively by the cardiac and respiratory activity sensor, skin resistivity sensor and body core acceleration sensor.

[1] ISO/IEC 20922:2016 standard – Message Queuing Telemetry Transport (MQTT) v3.1.1.

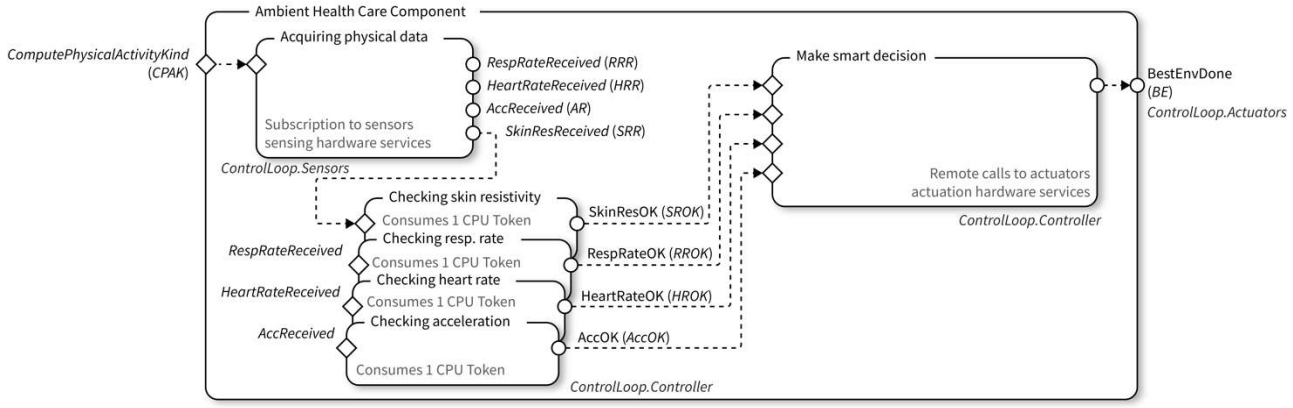

Figure 3. Ambient health care component GSM-based lifecycle

If the ambient health care component is already subscribed to the sensing hardware services of the physiological sensors, this stage just waits for the availability of new data in the gateway reception buffer. If a sensor stream is interrupted, this stage tries to reconnect to the relevant sensor. This stage is triggered using a command, introduced as "*ComputePhysicalActivityKind*," and it triggers four guards upon reception of new data, one guard for each sensed physiological value. The next four stages are grouped and are dedicated to sensor data correctness verification (i.e. heart rate data, respiratory rate data, skin resistivity data and acceleration data). These stages are triggered upon arrival off new data from the sensors, and milestones associated with the stages are set when data was proved to be correct. Operating theses stages causes the consumption of a CPU token, representing the available computational capabilities. The 6th and final stage represents high-level data processing, which uses sensor data in order to compute the environment for the user, and proceed with remote actuation hardware services calls to the relevant actuators. This service, because of the potentially heavy calculation it requires, can be executed on remote server accessed using traditional Internet-dedicated protocols. This best environment can thus be computed accounting for each user requirements (less stressful environment for cardiac risk patients, best recovering environment if the user performed exceptional physical activity during the day, etc.).

This lifecycle makes thus use of three services implemented in three different locations of the system: the data acquisition services are located in the sensors, the data verification service is in the gateway, and the remote Internet-based optimal environment computation service on a remote server. However, the resulting composition is implemented in the gateway, and the verification must provide guarantees over the presence of the relevant resources in order to fulfill the target system goal (i.e., the computation of the optimal living environment for the smart-home user).

In summary, the desired composition must make use of the physiological data acquired by the different sensors to compute the best physical environment and the best diet for a target smart-home user. The GSM representation of such composition is given in Figure 3. This composite GSM is implemented in the smart-home gateway, and guards and milestones highlighted in italic denote external events.

Finally, the artifact model is augmented with a feedback control loop describing the interaction between the sensors, actuators, controllers and plants used in our system. In our use case, the sensors of the feedback control loop correspond to all the wearable sensors worn by the smart-home user, the actuators correspond to all the environmental actuators available in the smart-home (e.g. blinds, light actuators, etc.). The controller represents the computing capabilities embedded in the gateway, while the plant of the feedback control loop represents the human-body as the system to be controlled.

While artifact and GSM-based model for s of lifecycle enable easy end-user specification of composition, the overall composition must be verified in order to guarantee that it is implementable and that safely executable. In order to deal with this challenge, we provide a formal representation of the GSM-based lifecycle with linear-logic axioms and formulas.

3. VERIFIABLE COMPOSITION

3.1 Linear Logic-based Smart Object Composition

Linear logic was introduced by Jean-Yves Girard in 1987 as an improvement over the classic logic that keeps track of resource consumption [9]. Indeed, while the classic logic focuses on the correctness of the demonstration, the linear logic adds a resource management aspect by interpreting propositions as consumable resources.

In the context of Web service composition, several segments of the linear logic have been studied: Rao et al. have used the intuitionistic multiplicative-additive fragment of the linear logic [17], while Papapanagiotou et al. studied the one-sided classical logic because of its formal equivalency to pi-calculus [14, 15]. However, the contributions behind both approaches are the same: representing Web services as linear-logic axioms, that will be used in the sequent calculus in order to formally verify the composition, but they lack physical resources modelization.

The main advantage of the linear logic when compared with the first-order logic relies on logical connectives than can be interpreted from a resource management perspective. The typical example is the statement $A \otimes B \vdash C$. In linear logic, this statement can be interpreted as "the production of resource C consumes resource A and resource B simultaneously". In the IoT context, with strong limitations on hardware resources such as battery or computing capabilities as well as limitations from a software perspective, with a limited amount of software component inputs and outputs, the ability of representing resources consumption is extremely useful, which makes the linear logic a tool of prime interest to model and verify IoT-based systems.

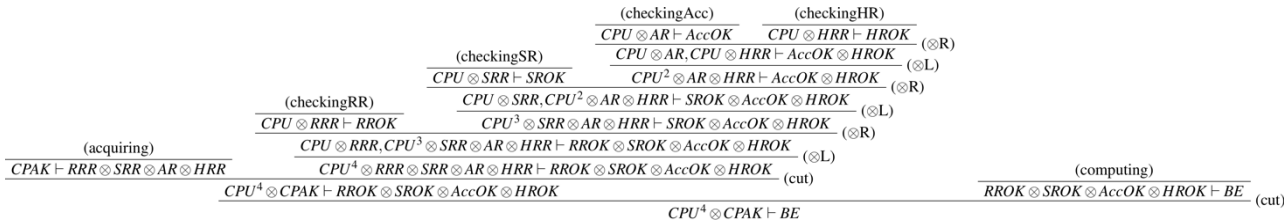

Figure 5. Proof tree of the ambient health care component

In order to prove correctness of the composite connected object lifecycle, we use the multiplicative conjunction ($A \otimes B$), thus reducing ourselves to the intuitive multiplicative segment of the intuitionistic linear logic (MILL). The rules of sequent calculus related to the MILL is given in Figure 4. Roughly speaking, if a proof tree exists for a given request (also called theorem) such as the lifecycle expressed in the linear logic, the composition expressed in terms of the lifecycle, will be considered verified. The construction of such proof trees consists of applying the set of rules to a linear logic statements until a known atomic statement is found. The composition can then be extracted from the proof, which carries a description of how the different sub-stages are executed.

$$\frac{}{A \vdash A}\ (\text{axiom}) \qquad \frac{\Gamma \vdash A \qquad \Delta, A \vdash C}{\Gamma, \Delta \vdash C}\ (\text{cut})$$

$$\frac{}{\vdash 1}\ (1R) \qquad \frac{\Gamma \vdash C}{\Gamma, 1 \vdash C}\ (1L)$$

$$\frac{\Gamma \vdash A \qquad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B}\ (\otimes R) \qquad \frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C}\ (\otimes L)$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B}\ (\multimap R) \qquad \frac{\Gamma \vdash A \qquad \Delta, B \vdash C}{\Gamma, \Delta, A \multimap B \vdash C}\ (\multimap L)$$

Figure 4. MILL inference rules

The linear logic is thus used to specify stages, since their opening guards and closing milestones depend on sentries. We propose to model the GSM-based lifecycle stages as the following formula:

StageName: *Guard* ⊢ *Milestone*

If several guards are available, they will be expressed using the multiplicative conjunction operator, and several milestones will be expressed in the same way. If hardware resources are consumed during the stage, they are also expressed using the multiplicative conjunction operator. Stages expressed in the linear logic will be thus considered as hypothesis. The outer stage, resulting of the composition of several sub-stages can also be expressed using similar formula.

For example, the hypothesis of our case study can be expressed as:

acquiring: $CPAK \vdash RRR \otimes HRR \otimes AR \otimes SRR$
checkingSR: $CPU \otimes SRR \vdash SROK$
chekingRR: $CPU \otimes RRR \vdash RROK$
chekingAcc: $CPU \otimes AR \vdash AccOK$
chekingHR: $CPU \otimes HRR \vdash HROK$
computing: $RROK \otimes AccOK \otimes SROK \otimes HROK \vdash BE$

And the composite or the outer stage to verify its correctness is expressed as the following:

$$CPU^4 \otimes CPAK \vdash BE$$

For conciseness purposes, all guards and milestones are abbreviated. However, the correspondence between abbreviations and the original terms can easily be determined through the stage name. For instance, if we consider the "*Acquiring physical data*" stage, we can see that the variable *CPAK* in the relevant linear logic statement corresponds to the *ComputePhysicalActivityKind* guard, while the RRR and all the other linear logic variables on the right-hand side corresponds to the *RespRateReceived* milestone.

The *CPU* variable represents a CPU load token, representing CPU usage by the data analysis tasks (for instance, the task will require 10% of the CPU computing capabilities). For example, the statement $CPU \otimes SRR \vdash SROK$ means that in order to confirm the correctness of the skin resistivity value, the stage needs the skin resistivity value and 10% of the computing capabilities of the gateway. Indeed, in linear logic, the sequent representation denotes resources consumption, and the multiplicative conjunction of resources on the left-hand side of a sequent mean both resources will be consumed simultaneously in order to provide the right-hand side of the sequent. Please note that CPU^4 denotes the multiplicative conjunction of 4 CPU variable occurrences.

The formal verification of the global composition is thus reduced to the proof of the outer stage statement. The proof of such statement can be realized using the interactive theorem prover Coq[2] augmented with a shallow embedding of the intuitionistic linear logic[3]. However, the embedding of the linear logic in such tools is out of scope of this paper. Interactive theorem proving provides means to illustrate all the proof steps, and provides indications about the correctness of each of the proof steps.

The proof tree corresponding the GSM-based lifecycle (i.e., its outer stage) is illustrated in Figure 5. This proof tree was built using a backward chaining approach, where the final statement is taken as a starting point. The rules used for each step of the sequent based calculus are indicated with reference to the set of rules in Figure 4.

One can note that each of the rule can be mapped to a GSM-related action. For instance, the cut rule denotes the serial composition of two (or more) stages: the last cut rule to be applied denotes the serial composition of the computing stage and all the data analysis stages. The $\otimes R$ and $\otimes L$ rules correspond to the parallel composition of several stages, and it is illustrated in our use case with the parallel composition of all the data analysis stages. Serial and parallel composition correspondences can be used to perform a posteriori analysis of the proof tree, and to deduce higher-level non-functional properties (a typical example would be the computation of the overall response time considering the influence of parallel or serial composition of atomic services on the resulting composition response time).

The proof tree holds for the verification of the targeted composition, because it demonstrates that the final statement can be obtained using only hypothesis defined. This gives a formal

[2] Available at: https://coq.inria.fr

[3] Available at: https://perso.ens-lyon.fr/olivier.laurent/l2coq.tgz

guarantee that the composite systems will behave according to the provided graphical specification.

4. RELATED WORKS

Considering the relatively wide scope of our work, we identify several communities have handled similar research problems and challenges. The first identified relevant research community is model-based architectures and development. When it comes to model-driven development, several attempts to unify modelization languages and frameworks can be found. One of these attempts is the GEMOC initiative, which pushes towards better integration and coordination of heterogeneous modeling languages [7]. The initiative provides an open-source tool, GEMOC Studio, which can be embedded with a variety of software and hardware modeling languages. In the context of the IoT, it is worth noting that this tool was used to model the widely known Arduino hardware and software platform [5]. However, our main reserve about this framework is that it requires extensive work to model different software or hardware architectures, which is a strong argument against the use of this framework under an IoT perspective, considering the strong heterogeneity of IoT devices, where different wireless protocols, CPU architectures or more global hardware architectures must be integrated into wider-scale systems. We also believe that diversity in modeling language is not the right perspective in the Internet-of-Things domain, where research should be focused on the elaboration of a generic modeling language that must be able to accurately represent the specificity of each devices, rather than the use of divers but coordinating modeling languages. This improves considerably the modeling experience, as the modeling process occurs using a unique modeling language. We thus aimed at the specification of a much simpler but more generic framework, allowing for both lightweitghness and cross-architecture deployment, with resources concerns directly embedded into the framework. Another challenge when considering the modelization of connected objects is the representation of the lifecycle of the objects. The model-driven software engineering community proposed the PauWare engine [1], a tool offering the specification and simulation of state chart XML and UML states machines. Both these modeling languages are representing software system reactions to various external and internal signals. However, this framework is strongly software oriented, and does not allow for precise resources modelization. Moreover, lifecycles are not declarative, and modification of a previously defined lifecycle is often a tedious process. Because the goal of the Internet of Things is to empower the end users and allowing them to control their environment that might be changing, having a flexible lifecycle model is a main priority. This is why we propose to use a more declarative approach for lifecycle specification, allowing for easier modification.

Because the work presented in this paper is related to the composition of connected objects seen as services, the contributions of the service-oriented computing community related to services composition must be studied. As mentioned earlier in our contribution, service-oriented architecture enables the construction of software systems using modular, reusable, interoperable and self-contained software components [16]. The advantages this architectural vision brought in terms of modularity and interoperability caused service-oriented architectures to be the principal motor of the development of the Internet and Internet-based services. Services composition describes the combination of several services in order to achieved a targeted functional goal, with respect to desired non-functional properties (such as response time, global availability, etc.). Formal verification of services composition is a widely studied research topic, and several mathematical formalisms were used in this context [2]: traditional automata-based representations [3], Petri nets modelization of services [10] or process algebras such as π-calculus [12]. However, these composition formalization struggle when it comes to the accurate resources representation of the connected objects. This can be solved using linear logic, and its embedded ability to represent resources consumption [9]. Contributions used linear logic in the context of web service composition: Rao used intuitionistic linear logic to model and verify resource-aware service composition in [12, 17], while Papapanagiotou used classical linear logic and its proven equivalence to the π-calculus process algebra to model and verify the resource aware service composition [14, 15]. Zhao used intuitionistic linear logic in the more restrictive context of RESTful web service composition [18, 19], and other contributions related to linear-logic based web-service composition were proposed [8]. However, the main problem when considering these contributions in the context of the Internet-of-Things is that they are not considered in addition to a formal and well-studied service model (at the exception of [18, 19], which rely upon the widely used RESTful service model).

Our contribution differentiation to the state of the art is twofold: the first differentiation point resides in the definition of a generic model enabling the representation of both the digital and physical nature of connected objects and composite connected objects, augmented with representations of their evolution over time (i.e., use lifecycle) and their closed feedback loop behavior. The second differentiating point comes with the direct mapping of the lifecycle model of the connected object to intuitionistic linear logic, enabling the verification of the overall composite connected object.

5. CONCLUSION

In this paper, we presented a smart object component-based model to build smart connected components using composition techniques in a similar way to Web service composition. The smart object model is based on an extension of artifact types and describes simultaneously its structure and its behavior. Behavioral Modeling was introduced using guard-stage-milestone lifecycle models. Additionally, we present a formal specification based on linear logic, not only to express consumable resources such as battery or computing capabilities, but also to automate connected objects composition from logical proof trees.

A promising research direction for our verification system would be the integration of automated proof search to produce the composition proof trees. However, while numerous tools of automated proof search for the first-order logic (such as E[4] or ACL2[5]) there is a lack of formal automated proof tools for the linear logic. The development of proof automation tools is thus a promising research direction in order to enable easier verification of wide-scale composite systems.

Acknowledgements: this work is generously supported by the COOPERA funding program of the Auvergne Rhône-Alpes.

6. REFERENCES

[1] Ballagny, C. et al. 2007. Endowing PauWare Components with Autonomic Capabilities. *Proceedings of the 1st Workshop on Model-driven Software Adaptation* (Berlin, Germany, Jul. 2007), 55–60.

[2] ter Beek, M. et al. 2007. Web Service Composition Approaches: From Industrial Standards to Formal Methods. *Proceedings of the 2nd International Conference on Internet and Web Applications and Services* (Morne, Mauritius, May 2007), 15–21.

[4] Available at: http://www.eprover.org

[5] Available at: http://www.cs.utexas.edu/users/moore/acl2/

[3] Berardi, D. et al. 2005. Automatic Composition of Transition-Based Semantic Web Services with Messaging. *Proceedings of the 31st International Conference on Very Large Data Bases* (Trento, Italy, Sep. 2005), 613–624.

[4] Bhattacharya, K. et al. 2007. Towards Formal Analysis of Artifact-Centric Business Process Models. *Business Process Management*. G. Alonso et al., eds. Springer Berlin Heidelberg. 288–304.

[5] Bousse, E. et al. 2016. Execution Framework of the GEMOC Studio (Tool Demo). *Proceedings of the 9th International Conference on Software Language Engineering* (Amsterdam, Netherlands, Oct. 2016), 84–89.

[6] Cerwall, P. 2017. Ericsson Mobility Report.

[7] Combemale, B. et al. 2013. Report on the First Workshop on the Globalization of Modeling Languages. *Proceedings of the 1st International Workshop on the Globalization of Modeling Languages* (Miami, Florida, USA, Sep. 2013), 3–13.

[8] Deng, J. et al. 2014. Complement Service Composition through Domain Template and Requirement Context. *Proceedings of the 11th International Conference on e-Business Engineering* (Guangzhou, China, Nov. 2014), 320–325.

[9] Girard, J.-Y. 1987. Linear Logic. *Theoretical Computer Science*. 50, 1 (1987), 1–101.

[10] Hamadi, R. and Benatallah, B. 2003. A Petri Net-Based Model for Web Service Composition. *Proceedings of the 14th Australasian Database Conference* (Adelaide, Australia, Feb. 2003), 191–200.

[11] Hull, R. et al. 2011. Introducing the Guard-Stage-Milestone Approach for Specifying Business Entity Lifecycles. *Web Services and Formal Methods*. M. Bravetti and T. Bultan, eds. Springer Berlin Heidelberg. 1–24.

[12] Jinghai Rao et al. 2004. Logic-Based Web Services Composition: from Service Description to Process Model. *Proceedings of the 2nd International Conference on Web Services* (San Diego, California, USA, Jul. 2004), 446–453.

[13] Nigam, A. and Caswell, N.S. 2003. Business Artifacts: An Approach to Operational Specification. *IBM Systems Journal*. 42, 3 (2003), 428–445.

[14] Papapanagiotou, P. et al. 2012. Diagrammatically-Driven Formal Verification of Web-Services Composition. *Diagrammatic Representation and Inference*. P. Cox et al., eds. Springer Berlin Heidelberg. 241–255.

[15] Papapanagiotou, P. and Fleuriot, J. 2011. Formal Verification of Web Services Composition Using Linear Logic and the Pi-Calculus. *Proceedings of the 9th European Conference on Web Services* (Lugano, Switzerland, Sep. 2011), 31–38.

[16] Papazoglou, M.P. 2003. Service-Oriented Computing: Concepts, Characteristics and Directions. *Proceedings of the 4th International Conference on Web Information Systems Engineering* (Rome, Italy, Dec. 2003), 3–12.

[17] Rao, J. et al. 2003. Application of Linear Logic to Web Service Composition. *Proceedings of the 1st International Conference on Web Services* (Las Vegas, Nevada, USA, Jun. 2003), 3–9.

[18] Zhao, X. et al. 2011. A Two-Stage RESTful Web Service Composition Method Based on Linear Logic. *Proceedings of the 9th European Conference on Web Services* (Lugano, Switzerland, Sep. 2011), 39–46.

[19] Zhao, X. et al. 2011. RESTful Web Service Composition: Extracting a Process Model from Linear Logic Theorem Proving. *Proceedings of the 7th International Conference on Next Generation Web Services Practices* (Salamanca, Spain, Oct. 2011), 398–403.